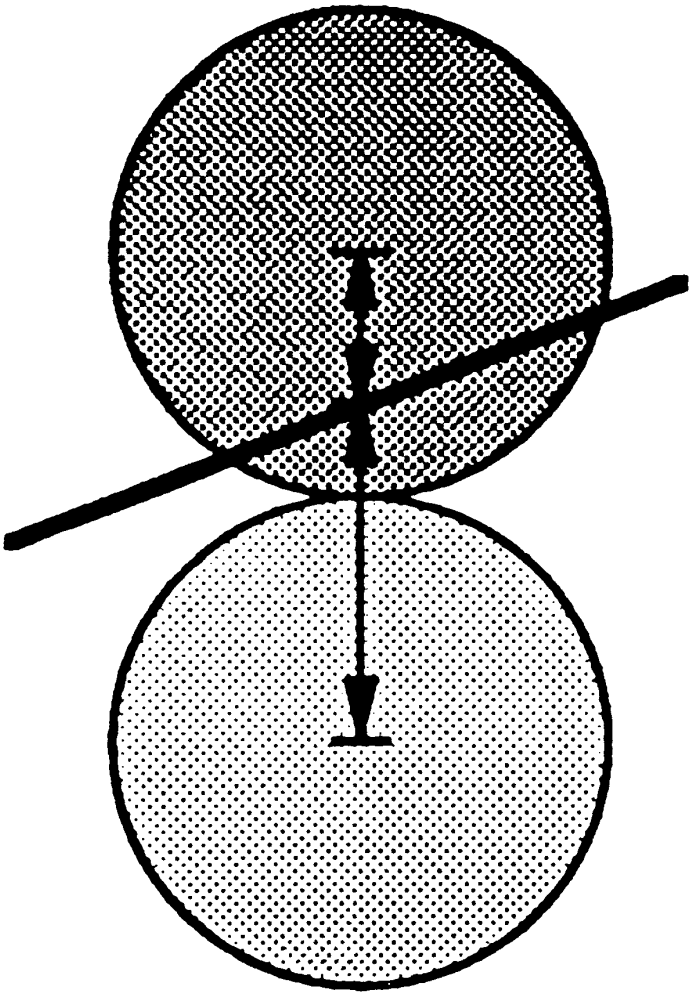
Superworkstation

The Silicon Graphics 4D/240GTX Superworkstation

Kurt Akeley
Silicon Graphics



A new class of machines, known both as superworkstations and graphics supercomputers, has become available. These workstations provide computing performance previously available only on supercomputers, and graphics performance at levels never before available. Their spectacular graphics performance is due both to advances in hardware technology and to tight integration between graphics and CPU subsystems.

Although superworkstation architectures share a commitment to graphics performance, and all evidence tightly integrated graphics subsystems, they differ fundamentally in their approaches to the partitioning of graphics-related computations, specifically in their support of the transformation process. These different partitionings, while seemingly new, are in fact well known options to designers of high-performance graphics systems.

This article presents a model that differentiates graphics architectures on the basis of graphics computation partitioning, and places superworkstation architectures within this framework. It then presents the reasoning that resulted in the architecture of the Silicon Graphics 4D/240GTX superworkstation. Finally, the architecture and performance of the 4D/240GTX are discussed.

Both traditional and contemporary high-performance graphics systems use parallelism to achieve their performance goals.[1-3] Fortunately, graphics computations are easily partitioned into functional tasks, which can then be assigned to multiple processors.

0272-17-16/89/0700-0071$01.00 ©1989 IEEE

Graphics task partitioning

All programmable, geometry-based raster-graphics systems implement five fundamental tasks:

1. *Generation.* Generate (create, acquire, or modify) graphics data. Organize these data into a structure.
2. *Traversal.* Traverse the graphics data structure, delivering graphics data to the appropriate processor(s).
3. *Transformation.* Transform graphics data from object-space coordinates into eye-space coordinates. Do whatever shading operations are appropriate in eye-space. (Lighting calculations are an example.) Transform eye-space coordinates to clip-space, clip these coordinates, and project the resulting coordinates to screen-space.
4. *Scan conversion.* Render the resulting screen-space objects (points, lines, and polygons) into raster memory. Iterate parameters as appropriate throughout this process. Do whatever shading operations are appropriate in screen-space.
5. *Display.* Scan the resulting pixels out of raster memory to a display monitor.

Because these tasks are executed in order (*transformation* precedes *scan conversion,* etc.) they are always implemented as a pipeline of processes. The choice of which tasks are executed on the application-programmable CPU, and which on special-purpose graphics processors, differentiates high-performance graphics architectures.

Display is a simple but performance-sensitive task. It is simple and regular because raster data are always transferred to the monitor in scan-line order, whether from a single raster or from a small set of subscreen rasters. It is performance-sensitive because monitors must be provided complete image data at least 60 times per second if irritating flicker is to be avoided. Thus a 1280 × 1024-pixel color monitor, typical of superworkstations, demands data at the sustained rate of 236 Mbytes per second (1280 × 1024 pixels per frame × 60 frames per second × 3 bytes per pixel). While this data rate would significantly burden any general-purpose processor, its regularity allows support to be provided in the form of a simple, special-purpose processor. For these reasons the *display* task is always executed on such a processor. (Systems that scan convert into nondisplayable memory require an additional copy operation. This copy may be supported by an application-programmable processor.)

Generation, while more forgiving of performance variation, is demanding in its algorithmic complexity. Interactive simulations, for example, generate graphics data by culling and selecting object models based on viewer position and direction. Stress analysis requires that new colors be generated for each vertex in a mechanical model. Position and length of each bond are changed in the model of a molecule. Multiple windows, each with its own specific requirements, are maintained simultaneously. Because generation tasks are complex, widely varied, and application-specific, they are always executed on the application-programmable processor.

Traversal, transformation, and scan conversion, however, have all been successfully implemented both on application-programmable processors and on special-purpose processors. Some examples will clarify these alternatives.

Architecture examples

Bitmap workstations such as the Xerox Alto[4] and the original Sun systems completely supported generation, traversal, and transformation (typically in just two dimensions), with their application-programmable processors. While some hardware support was provided to increase the performance of scan conversion (BitBlt microcode on the Alto, a 32-bit barrel shifter on the Sun workstation), this hardware was driven directly by code executing on the application processor. Only the display task executed on a separate, special-purpose processor. We refer to such an architecture as GTXS-D, indicating that generation, traversal, transformation, and scan conversion (GTXS-) execute on the application-programmable processor, display (-D) on a separate, special-purpose processor.

Traditional host-terminal implementations, on the other hand, offer a very different assignment of graphics tasks. The DEC VAX/E&S PS-390 combination, for example, supports only the generation task on the application-programmable VAX processor. Traversal, transformation, scan conversion, and display are all executed on special-purpose processors within the PS-390 terminal. While some of the PS-390 processors are user programmable, their special-purpose design clearly distinguishes them from the general-purpose, application-programmable VAXs. We refer to such host-terminal architectures as G-TXSD, indicating that only *generation* executes on the application-programmable processor.

Many contemporary systems, including superworkstations, shun the architectural extremes of GTXS-D and G-TXSD. Graphics accelerator boards that include display, scan conversion, and sometimes transformation processors are widely available for machines ranging from personal computers to engineering workstations. With the addition of such accelerators, GTXS-D systems are modified to become either GTX-SD or GT-XSD systems.

While superworkstations have sufficient application-programmable CPU power to generate, traverse, transform, and scan convert with surprisingly high performance, all currently support one or more of these tasks with special-purpose processors. Superworkstations available from Apollo, Ardent, and Stellar implement variations of GTX-SD architectures. All Silicon Graphics workstations, including the 4D/240GTX superworkstation, are variations of GT-XSD architectures.

The assignment of graphics tasks to special-purpose processors is a compromise of such factors as cost-performance, generality, configurability, and user coding style. Before investigating the merits of various partitionings as they pertain to superworkstations, we simply observe that all of the alternatives have legitimate application. Examples of each recur as graphics architectures evolve.[5]

The Silicon Graphics task partitioning

As we have seen, the issue facing a graphics system designer is not *whether* to partition graphics tasks among various processors, but rather *how* to partition these tasks. Our choice to support generation and traversal on our application-programmable CPU and transformation, scan conversion, and display in dedicated hardware was made six years ago when our first workstation product was developed. Since that time the internal details of both the CPU and of the dedicated graphics systems have changed dramatically. Application-programmable CPUs have gone from 0.5 MIPS CISC machines (68010) to 20 MIPS RISC machines (MIPS R3000), and from single to multiprocessor organizations. Dedicated graphics processors have evolved from hardwired engines (such as the original Geometry Engine) to the 40-Mflop (sustained) programmable floating-point subsystem of the 4D/240GTX. But the fundamental GT-XSD task partition has remained unchanged.

What are the characteristics of GT-XSD task partitioning that allow us, and in fact encourage us, to continue to design with it? We consider both the options of increasing (G-TXSD) and of decreasing (GTX-SD) the number of graphics tasks executed on special-purpose processors.

Why not traverse in hardware?

The arguments against a G-TXSD partition boil down to just one issue: immediate mode graphics. Moving the traversal task from the application-programmable environment of the general-purpose CPU into dedicated graphics hardware offers the promise of increased performance, but at the price of imposing hardwired traversal algorithms, predefined data structures, and clumsy modeling interaction. When the generation task is ignored (as it often is in graphics demonstration programs), machines with G-TXSD architectures shine. Their inflexible traversals and costly model changes do not affect performance, because simple traversals are adequate, and the model is not being changed. Thus host-terminal G-TXSD systems can display complex models with rapid viewpoint changes, but offer limited support of either user imposed or computed changes to the model being viewed.

PHIGS and PHIGS+ are easily and efficiently implemented on a GT-XSD platform.

The VAX/PS-390 combination, often used in molecular modeling applications, offers a suitable example. Users of this system quickly learn that, while molecule stick figures can be quickly viewed from any direction, and in fact wiggled to improve depth perception, changes to the structure of the molecule are not interactive. Thus individual molecules can be viewed, but studies of the interaction of neighboring molecules are limited.

It is sometimes thought that the limitations of host-terminal systems are the result only of insufficient bandwidth on the thin wire connecting the host to the terminal. While low bandwidth does reduce interactivity, all G-TXSD systems, whether connected by

an RS-232 line or a 500 Mbyte-per-second backplane, remain limited by predefined traversal algorithms, fixed data structures, and clumsy, often slow, modifications to geometry. Only when generation and traversal execute on the same application-programmable processor (or processors) are these limitations overcome.

Immediate mode graphics is graphics-driven entirely, i.e., generated and traversed, by an application program executing on the programmable CPU. While it imposes additional requirements on the dedicated transformation graphics hardware, it frees the graphics programmer from the constraints of predefined display lists, display-list traversal, and inefficient change mechanisms to model definitions. The generation task of an immediate mode molecular modeling program, for example, computes bond lengths and angles resulting from an interactive condition, then changes them by simply editing its own data structures. The subsequent traversal takes its data from the shared data structures, effecting the changes to the image automatically. Alternately, a real-time simulation traversal can modify its own operation, culling and making model detail decisions as it operates based on the actual time elapsed during the construction of a frame. A generation task could make these decisions only in the abstract, based on viewer position and direction. Ultimately, an immediate mode workstation performs well not only on canonical demo programs, but also on interactive applications, applications that change both model data and traversal as a function of user input.

Before leaving the issue of application-driven traversal, it is important to consider the PHIGS graphics standard. PHIGS, and its proposed extension PHIGS+, define a retained-model graphics standard. Traversal of the graphics data retained in PHIGS structures is defined by the standard, and is therefore not under direct application control. Thus PHIGS offers the software architectural equivalent of a G-TXSD hardware architecture. Two points must be made:

1. Because PHIGS inherits the limitations of G-TXSD architectures, a high-performance system should offer an immediate mode graphics interface in addition to its PHIGS/PHIGS+ interface.
2. PHIGS and PHIGS+ are easily and efficiently implemented on a GT-XSD platform. An immediate mode graphics interface, on the other hand, cannot be efficiently implemented on a G-TXDS platform.

Why transform in graphics hardware?

Tight coupling between model generation and traversal is critical to the performance of superworkstation machines; it is a characteristic that is shared by all superworkstation architectures. These architectures differ, however, in their implementation of the transformation task. What is the case for doing transformation in special-purpose processor hardware? We consider the following issues:

1. Impact of graphics requirements on CPU architecture.
2. Efficiency of hardware use during anticipated operation.
3. Configurability.
4. Feature/performance trade-off.

We will see that the first three considerations argue for special-purpose transformation processing, and that only the last argues against it.

Impact on CPU architecture

The impact of including the traversal process on the general-purpose CPU is low. The CPU must be able to traverse data structures and transfer data at the rate that they are consumed by the transformation process. While this rate is substantial in a modern workstation, and some hardware support in the CPU may be required to accommodate it, there is no need to alter the organization of the chosen CPU radically.

Transformation, on the other hand, makes tremendous demands of its supporting hardware. The 4D/240GTX, for example, executes approximately 100 floating-point operations per transformed vertex, which it generates at the rate of 400,000 per second. Thus its dedicated transformation engine operates at a sustained rate of 40,000,000 floating-point operations per second (40 Mflops). This computation rate is simply not achievable using currently available general-purpose CPU technology. If it is to be sustained, either a special-purpose processor must be designed to support it (for example, a Geometry Engine), or the application-programmable CPU must itself be designed as a special-purpose system (for example, a vector processor). Such a special-purpose CPU will compromise performance across the wide variety of nongraphics application programs. It may also become separated from the mainstream of CPU technology, increasing the difficulty of tracking future hardware advances.

Efficiency of hardware use

It would seem that cost-performance is optimized when no hardware in the machine goes unused. Thus dedicated graphics hardware, when graphics computations are not being performed, is a poorly used resource. However, cost-performance is also reduced when hardware systems are underused. A single-precision graphics Mflops can be supported in special-purpose hardware at a substantially lower cost than it can be in the (typically double-precision) floating-point hardware of an application-programmable processor. The ratio on the 4D/240GTX, for example, is greater than 10:1. Thus, while graphics hardware in a GT-XSD machine is poorly used when graphics computations are not being performed, CPU hardware in a GTX-SD machine is poorly used when graphics computations *are* being done. Again, the choice is not whether to compromise, but how.

GTX-SD workstations compromise simultaneous computation and graphics performance because the graphics transformation consumes a disproportionate, often total, fraction of the available application processor(s). When computation and graphics performances are specified independently, they create a misleading impression of GTX-SD interactive capability. The GT-XSD 4D/240GTX, on the other hand, efficiently executes computationally intensive applications and real-time 3D geometric graphics simultaneously. Traversal at full graphics performance consumes only a fraction of the power of one of its four application processors. Thus over three-quarters of its CPU performance is available for generation and other computation tasks.

Stress analysis, for example, is a computationally intensive task. The 4D/240GTX can apply almost all of its CPU performance to this problem, use its immediate mode graphics capability to update the display model, and support simultaneous viewing of a constantly recolored image.

Configurability

Because only a small fraction of the 4D/240GTX CPU power is required to support full-performance graphics, its graphics and CPU performance are effectively decoupled. Simply removing the special-purpose graphics processors creates a powerful and efficient application engine (4D/240S). Alternatively, application processors can be removed (two-processor 4D/220GTX) or run at lower clock rates without affecting graphics performance.

The 16-MHz, two-processor 4D/120GTX, for example, provides a cost-effective solution in embedded simulation environments. While the generation task is typically shared between the 4D/120GTX and another controlling processor, reducing the computational requirements on the superworkstation, full graphics performance, including traversal, is still required.

Feature/performance trade-off

The disadvantage of special-purpose hardware is its reduced flexibility. We expect a more rigid feature/performance trade-off from a dedicated graphics processor than from an application-programmable processor. Indeed, for the same reasons that special-purpose application processors compromise performance when executing general-purpose algorithms, dedicated special-purpose processors also operate best when executing their intended algorithms. Also, unlike special-purpose application processors, dedicated processors are inconvenient, or simply impossible, to program from the application level. Thus a dedicated transformation processor must be provided by its manufacturer with a wide variety of graphics algorithms if it is to exhibit gradual feature/performance trade-offs.

The advantages of application-programmable traversal and immediate mode graphics compel us to keep the traversal process located on the CPU.

The single-precision floating-point transformation processor used in the 4D/240GTX, while not application programmable, is driven by RAM-based microcoded controllers. Each software release adds features and capabilities to its large initial set. Although its hardware design anticipated some specific requirements of the algorithms for which it was tuned, it has proved to be readily adaptable, and its performance on new and original algorithms is comparable.

Our recent implementation of nonuniform rational B-spline (NURBS)[6] surfaces provides a useful exam-

ple. NURBS generation is both algorithmically complex and computationally intensive. By implementing the (complex) trimming algorithms in software on the application-programmable processor, and the floating-point intensive mesh expansion in the special-purpose transformation processor, we achieved high performance on a feature that was not intended when the transformation processor was designed.

Architectural alternatives

The advantages of application-programmable traversal and immediate mode graphics compel us to keep the traversal process located on the CPU. We have chosen the advantages of CPU independence, cost-effective simultaneous computation and graphics operation, and configurability demonstrated by a dedicated transformation processor over the potential benefits of feature/performance trade-off exhibited by CPU transformation machines. This given, we have designed our dedicated transformation engine to deliver high performance across a broad range of graphics algorithms.

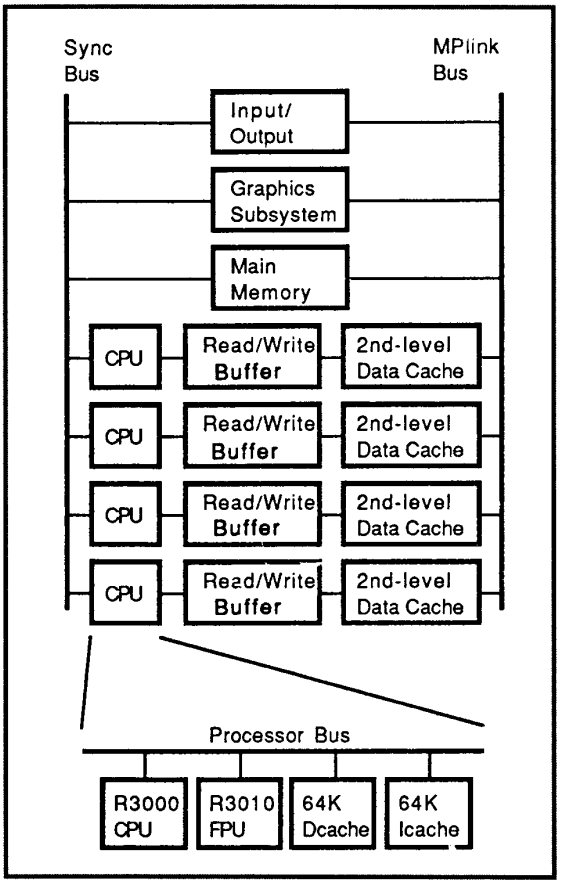


Figure 1. System block diagram.

Computing system architecture

As we saw previously, our implementation of a dedicated transformation engine allowed us to choose our CPU architecture without consideration of special-purpose graphics computation requirements. Given this freedom, we have implemented a parallel-scalar multiprocessor subsystem. This general-purpose architecture performs extremely well across a wide variety of both scalar and vector applications. Figure 1 is a block diagram of the major components of this subsystem.

Key to both the performance and the efficiency of the processor system is a hierarchy of buses, each tailored to a particular function. The sync bus provides high-speed synchronization between the four application processors in support of fine-grained parallelism. Processor buses support full-speed data and instruction transfer between individual processors and their first-level instruction and data caches. The MPlink bus protocols support consistent data sharing between processors, as well as high-speed block data transfer between processors, the memory and I/O subsystem, and the graphics subsystem. Processors communicate to the MPlink bus through individual read/write buffers and second-level data caches, both increasing the efficiency of MPlink bus use and supporting the data consistency operations.

Sync bus

The sync bus was designed to meet the synchronization needs of a multiprocessor supporting efficient fine-grained parallelism. Single-application programs can make efficient use of parallel processors at the individual loop level. Large-grain parallelism, such as consistent access of shared data structures between generation and traversal tasks, is supported as well.

The sync bus provides 64K individual test-and-set variables, which are accessed in the physical address space of each processor. These variables can be mapped directly into the virtual address space of a user process, supporting application synchronization, with overhead measured only in fractions of microseconds. As a result, many programming and

compiler techniques developed for vector processors are also suitable for the scalar multiprocessor system. For example, strip mining, the technique of taking a long vector and breaking it into a number of strips for a vector register, is implemented by providing data strips to each of the four processors.

Irix (the Silicon Graphics implementation of Unix V.3), is itself a parallel, fully-symmetric multiprocessing application. Several thousand test-and-set variables are used by the system to provide fine-grained locks for its control variables. Irix also benefits from the sync-bus-interrupt facility, which supports both directed and broadcast interrupts. The flexible interrupt distribution scheme allows process scheduling to support, rather than disrupt, the private first-level caches.

Processor bus

The RISC CPU, floating-point coprocessor (FPU), and first-level instruction and data caches of each processor are connected by the processor bus. Each 32-bit processor bus operates at twice the 25-MHz CPU clock rate, sustaining instruction and data transfer at up to 200 Mbytes per second. Because each CPU includes its own processor bus, the aggregate processor bus bandwidth scales directly with the number of CPUs in a system. Processor-to-cache transfer is sustained at up to 800 Mbytes per second in the four-processor 4D/240GTX, 400 Mbytes per second in the two-processor 4D/220GTX.

First-level instruction and data caches each store 64 Kbytes, resulting in a four-processor aggregate first-level cache size of 0.5 Mbytes. A read buffer and a four-word-deep write buffer isolate the first-level caches from the MPlink bus and its second-level data caches. In addition to improving performance, these buffers provide an asynchronous interface between the 25-MHz processors and the 16-MHz MPlink bus. Processor clock speed can be increased to improve performance, or decreased to reduce cost, independent of the system bus frequency.

MPlink bus

The MPlink bus supports the system's cache consistency protocol and provides block data transfer between the four processors, the memory and I/O subsystem, and the graphics subsystem. Second-level 256-Kbyte data caches, one associated with each processor, are best thought of as components of the MPlink bus. Operating as a single, synchronous unit, the four second-level data caches and the MPlink bus monitor each block data transfer. Together, using a convention known as the Illinois Protocol,[7] they ensure that only valid data are made available to the four processors.

Because bus monitoring is a single, synchronous operation, the 64-bit data path of the MPlink bus can be efficiently used. The peak data rate for this bus is 128 Mbytes per second (8 bytes per clock $\times$ 16 clocks per bus). A data rate of 64 Mbytes per second is sustained, implying an overhead of only 50 percent for both bus-monitoring and cache-line updating operations.

Graphics support

Each processor includes hardware to support efficient transfer of graphics data to the transformation engine. The transfer is referred to as a *three-way transfer* because

1. The application processor initiates the transfer, after it completes the virtual-to-physical address translation of the first word of a short (two-through-four) word data vector.
2. The memory subsystem provides the specified data.
3. The graphics system accepts the data.

While three-way transfer can be thought of as a DMA operation, its atomic nature (the three steps cannot be interrupted) and integral address translation make it far more efficient than a typical DMA implementation. We prefer to think of the short data vectors as additional data types for which the hardware has been tuned, just as double-precision floating-point numbers are constructed of two 32-bit words on many machines. A 3D coordinate, for example, is a vector of three floating-point values, representing *x*, *y*, and *z* in object space. Data types include 2D, 3D, and 4D coordinates, vertex normals (*nx,ny,nz*), and three- and four-component colors (r,g,b,α).

The three-way transfer hardware, which is added without modification to the RISC integrated circuits, allows the Silicon Graphics immediate mode graphics library to transfer randomly located vertex data at the rate of over 400,000 position/normal pairs per second. Graphics data can be stored in an application-defined structure and traversed with application code at the full transformation rate. Because graphics library subroutines are called indirectly through a shared library interface, programs compiled for the 4D/240GTX can be run on any 4D product without

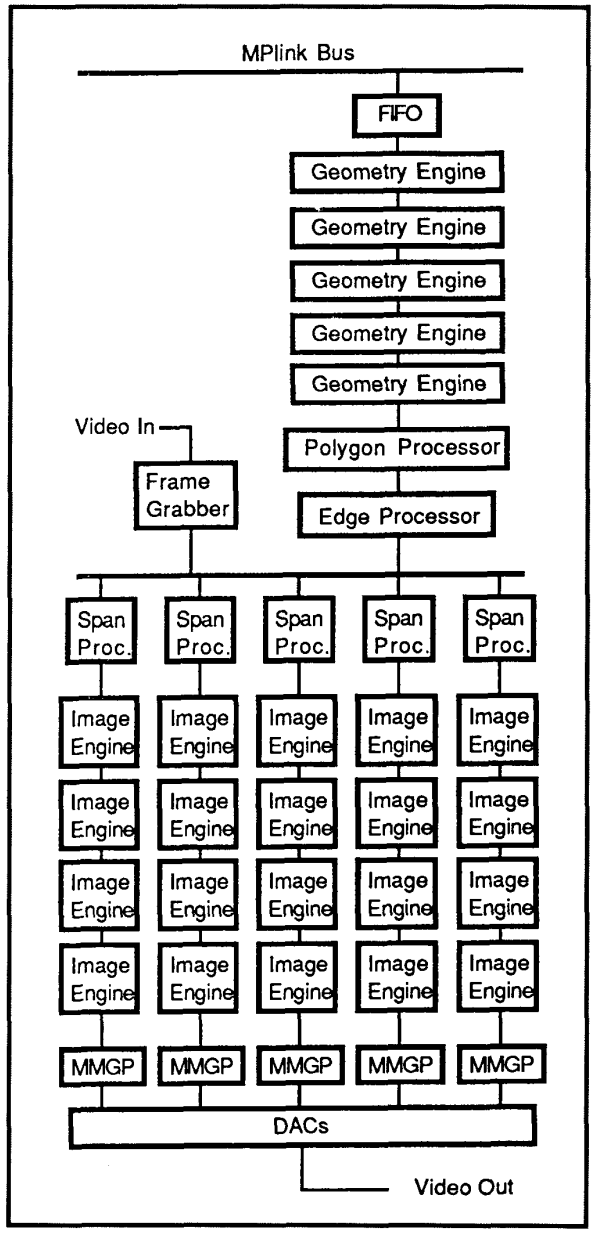


Figure 2. Graphics subsystem.

recompilation or relinking, and without performance loss.

Performance

MIPS R3000 CPU and R3010 FPU RISC components are used in each of the four processors. Running at 25 MHz, each of the four processors executes approximately 20 million instructions per second (MIPS), and achieves four double-precision Linpack Mflops. Their performance operating as a system, however, cannot be so easily reduced to two numbers.

One useful performance measure is throughput, the rate that n equivalent processes execute on an n-processor system. Both Dhrystone and Linpack[8] benchmark results document that the 4D/240GTX delivers near-linear throughput performance improvement.

	1 processor/ 1 program	4 processors/ 4 programs	Ratio
Dhrystone	37,400	149,085	3.98
(VAX MIPS)	21.3	84.8	
LINPACK	4.0	15.3	3.83

Linpack numbers are for 100 × 100 double-precision data, measured with coded BLAS. Dhrystone VAX MIPS are related to the base numbers by a factor of 1,760. Large, write-back second-level data caches, ample I/O capability, and careful attention to reduction of system overhead all contribute to near-linear throughput performance. The fully semaphored operating system, which executes as a distributed process rather than in master/slave mode, allows the four processors to make balanced throughput contributions. (Semaphores block less than 0.1 percent of the times they are encountered.)

Near-linear throughput performance, though certainly not achieved by all multiprocessor systems, is more easily accomplished than near-linear performance increase of a single process as it is distributed across the n-processor system. Unfortunately, the performance of a single application distributed across all available processors, a very significant measure of a multiprocessor system, is not well represented by benchmarks in use today. An indication of the performance delivered by the 4D/240GTX is provided by the table below.

	1 processor/ 1 program	4 processors/ 1 program	Ratio
LINPACK	6.6	16.6	2.56
BRL RTFM	1360	5034	3.7

Linpack numbers are for 1,000 × 1,000 double-precision data measured with coded BLAS. BRL RTFM is the Ballistic Research Lab's ray-tracing figure of merit, an application designed to test parallelization capability. Hardware cache consistency, distributed inter-

rupts, synchronization supported by a dedicated bus, and an automatic parallelizing compiler all contribute to the support of $1 \rightarrow n$ parallel execution.

Graphics architecture

The graphics system of the 4D/240GTX is itself partitioned into four subsystems, corresponding almost exactly to the task partitions described earlier in this article. Figure 2 is a block diagram of the entire graphics system.

Transformation subsystem

Vertex transformation is a very demanding task in an interactive 3D environment. Roughly 100 single-precision floating-point operations are required to transform, light, clip test, project, and map an object-space vertex to screen-space.[9]

Operation	FLOPs
Vertex transformation	28
Normal transformation	15
Single-source lighting	28
Clip testing	6
Projection	11
Map to screen-space	9
Total	97

The 4D/240GTX dedicated transformation subsystem, called the Geometry Engine, processes vertexes at the rate of 400,000 per second, sustaining a computation rate of 40 Mflops.

Careful attention to the flow of data and control in the design of the Geometry Engine ensures that polygonal and mesh boundaries do not substantially affect performance. Thus full transformation performance is available not only to large blocks of contiguous vertexes, but also to the vertexes of independent polygons and of short triangle meshes. Both independent triangles and independent quadrilaterals, for example, are drawn at the full 400,000-vertex-per-second rate (100K quadrilaterals per second, 135K triangles per second). Meshes of only five vertexes (three triangles) draw at over 90 percent of the peak triangle mesh rate (also 135K, see Figure 3).

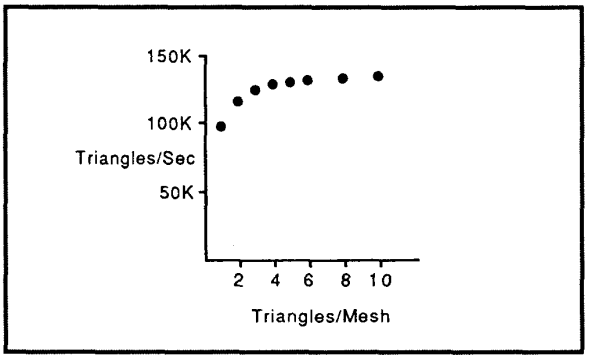


Figure 3. Triangle mesh performance.

A workstation that is to support immediate-mode graphics (with the desirable properties described in previous sections) must achieve full transformation performance on small data sets, as well as support both generation and traversal on its application-programmable processor. The transformation processor of the 4D/240GTX, with the three-way transfer hardware of the processor system, meets this requirement. Special-purpose application processors in GTX-SD machines sometimes do not.

The Geometry Engine comprises a single conversion and FIFO module, followed by five identical microcoded floating-point processors. The six processors are organized as a single pipeline. Each executes a specific subset of the standard transformation algorithm.

Coordinate and color data are accepted by the conversion module in single- and double-precision floating-point, integer, and packed-integer formats. All data are converted to single-precision IEEE floating-point format before being transferred to the first floating-point processor. A 512-word-deep FIFO precedes the conversion module. The processor that is currently executing the traversal process is interrupted whenever the FIFO becomes more than half full. It stalls until the FIFO empties past a low-water mark, then returns control to the application graphics program. Thus graphics flow-control adversely affects traversal performance only when the allowable transfer rate is exceeded.

The floating-point processors are identical modules, each consisting of a 20 Mflops peak data-path component (FPU) and a RAM-based microcode sequencer. Data and control transfer between the processors is arranged to avoid lost FPU cycles. Transformation tasks are distributed among the five processors as follows:

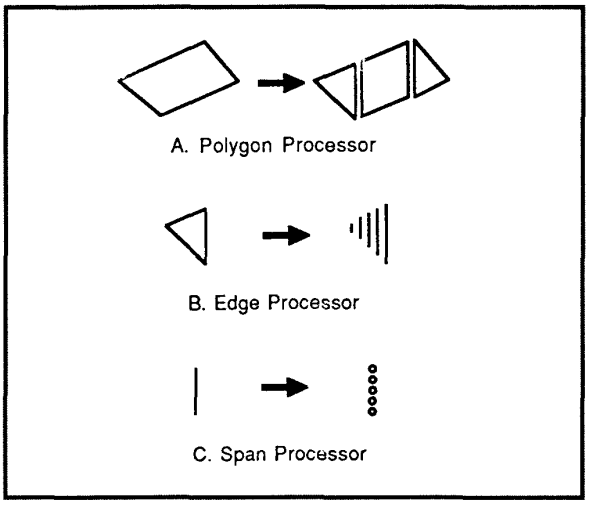


Figure 4. Polygon decomposition.

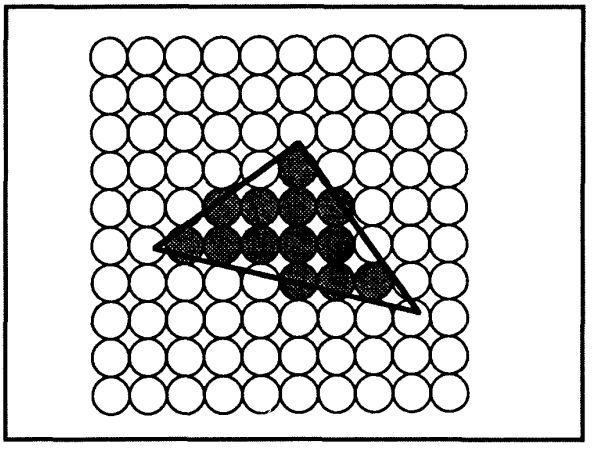


Figure 5. Point-sampled polygon.

1. Matrix and normal transformation. Coordinate matrix and normal (inverse-transpose) matrix stack maintenance. Normalization of vertex normals.
2. Lighting calculations.
3. Clip testing.
4. Perspective division. Clipping (when required).
5. Viewport transformation. Color clamping to a maximum value. Depth-cue calculations.

Scan-conversion subsystem

Output data of the transformation subsystem specify the vertexes of points, lines, and polygons in screen coordinates. The scan-conversion subsystem performs the calculations required to reduce these vertex data to individual pixels. Each pixel is assigned x, y, and z coordinates and r, g, b, and α color values. Color and z data are interpolated linearly between vertexes and between the edges of polygons.

Polygon scan conversion is partitioned across three separate processors within the scan-conversion subsystem. The polygon processor decomposes polygons into screen-aligned trapezoids whose left and right edges are vertical (see Figure 4a.) These trapezoids are themselves decomposed into vertical single-width spans by the edge processor (see Figure 4b). Finally, the individual spans are reduced to pixels by an array of five span processors, each of which operates on a single span (see Figure 4c).

Color and depth slopes are computed at each decomposition step, resulting in bilinear interpolation across the interior of the polygon.[10] The scan conversion is *point-sampled*, meaning that

1. Only pixels whose exact centers are within the precise polygon boundary are filled.
2. Color and depth parameters are iterated with subpixel precision, then corrected to correspond exactly to the (pixel-centered) sample points (see Figure 5).

As a result, iterated color and depth values remain planar across polygonal surfaces, avoiding unnecessary color banding and yielding clean z-buffer intersections. Also, adjacent polygons neither share nor omit pixels, properly supporting transparent renderings.

Polygon spans are generated at the rate of 2 million per second. Span processors reduce spans to pixels at the rate of 16 million pixels per second, resulting in an aggregate (five-span processors) fill rate of 80 Mpixels per second. Line pixels are generated by the edge processor at the rate of 16 Mpixels per second.

Raster subsystem

The raster subsystem contains 20 Image Engines, each of which is an independent state machine that controls 1/20 of the frame buffer memory. Groups of four Image Engines are driven by each span processor. The array of Image Engines tiles the frame buffer in a five-wide, four-high, pattern (see Figure 6). This 2D interleaved tiling pattern is arranged so that any screen-aligned 5×4 region is supported by all 20 Image Engines. As a result, the Image Engines operate in parallel even when very small regions are being filled.

Bitplane memory is organized into five banks, with a total of 96 bits per pixel. The banks are arranged as follows (see Figure 7):

- *Image banks.* Two banks of 32 bits each provide direct support of double-buffered images. Organized as eight bits each of red, green, blue, and alpha data when used for RGB (true color) imaging, and as 12-bits each when used for pseudocolor imaging.
- *Depth bank.* One bank of 24 bits stores 24-bit integer depth information when used with the Image Engine *z*-buffer pixel algorithm. It is also available for image data.
- *Overlay bank.* One bank of four bits, two bits of which are reserved for the window manager to display pop-up menus. The two remaining banks are available to application programs for both overlay and underlay images.
- *Window ID bank.* One bank of four bits, used by the window manager to tag pixels based on the drawing process to which they belong.

The four-bit window ID field identifies each pixel as belonging to one of (up to) 16 currently active windows. All pixel algorithms, except *high-speed clear*, compare the contents of this ID field to the ID of the currently drawing window process. If the IDs do not match, no change is made to the current pixel values. Thus both partially obscured and nonrectangular windows are supported with no performance penalty.

Image Engines operate as specialized memory controllers, supporting video RAM refresh, display refresh, and a handful of pixel access algorithms. These algorithms include

- *Replace.* Replace the destination color with the source color.
- *Z-buffer.* Compare the source and destination *z* values. If the test passes, replace the destination color and *z* with the source color and *z*. Any combination of greater than, equal to, or less than can be specified as the test condition.
- *Alpha blend.* Replace the destination color with a linear combination of the source and destination colors.[11] Each of the four color components is operated on with the following equation:

$$C_{dst} = F_{src}C_{src} + F_{dst}C_{dst}$$

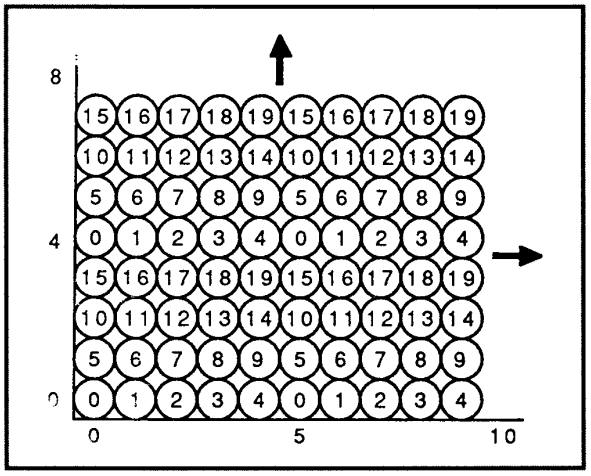


Figure 6. Image engine tiling pattern.

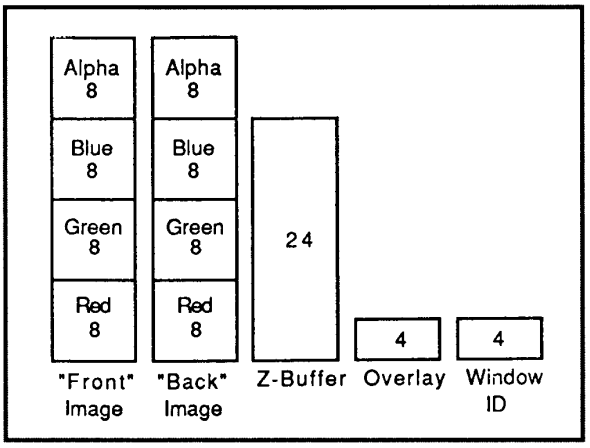


Figure 7. Bitplane assignments.

Applications choose the component multipliers from the following lists:

$$F_{src} \in 0,\ 1,\ \alpha_{src},\ 1-\alpha_{src},\ \alpha_{dst},\ 1-\alpha_{dst},\ C_{dst},\ 1-C_{dst}$$
$$F_{dst} \in 0,\ 1,\ \alpha_{src},\ 1-\alpha_{src},\ \alpha_{dst},\ 1-\alpha_{dst},\ C_{src},\ 1-C_{src}$$

The subscripts *src* and *dst* refer to the incoming and current color values respectively.

- High-speed clear means a simple replace is available only for large, screen-aligned rectangles.

Although the Image Engines are simple machines, their parallel operation and multiple algorithms result in extremely powerful pixel-fill operation. Aggre-

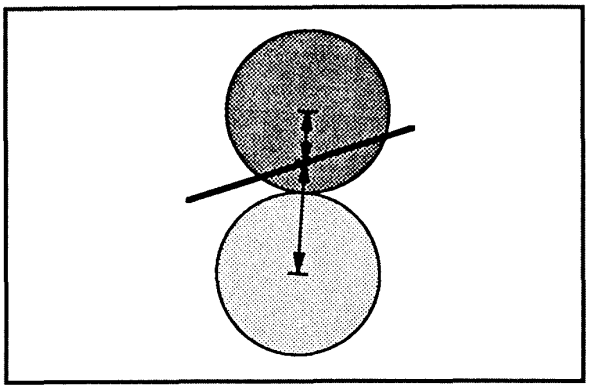


Figure 8. Antialiased line generation.

gate performance for the various pixel algorithms is as follows:

Pixel Algorithm	Fill Rate Mpixel/sec
Replace	80
Z-buffer	40
Alpha blend	16
High-speed clear	160

Display subsystem

The display subsystem receives pixel data from the frame buffer, interprets it, and routes the resulting red, green, blue, and alpha data to the digital-to-analog converters for display. Five multimode graphics processors (MMGPs) operate in parallel, one assigned to the pixels controlled by each span processor. These MMGPs receive all 64 image bank bits, the four auxiliary bank bits, and the four window ID bits for each pixel. They interpret the image and auxiliary bits as a function of the window ID bits, using an internal 16-entry table. Thus, windows can independently select single- or double-buffer operation, and double-buffer windows can swap buffers independently. Color index (pseudo color) and RGB (true color) operation are also selected independently on a per-window basis.

Window ID interpretation allows complete window support in a system that scan converts into and displays from the same raster memory (frame buffer). Eliminating the need for copy operations between raster memory and display memory reduces system bandwidth requirements, allows double-buffer windows to be easily synchronized with monitor refresh, and supports 30-Hz and even 60-Hz frame refreshes. Rapid frame rates are critical in real-time simulation environments, and they were an important consideration in the design of the 4D/240GTX.

Context switching

The graphics subsystem is designed to support context switching with minimal overhead. Because significant quantities of state are accumulated in the five floating-point processors of the Geometry Engine, each maintains complete context for 16 independent processes in its local data memory. The floating-point processors are also able to dump and restore context to and from a host processor, allowing more than 16 processes to share the hardware. Thus a *working set* of up to 16 processes is supported, with essentially no limit to the total number of processes.

Because the edge, span, and image processors are unable to return state information, the few states stored in these processors are shadowed by the polygon processor. The polygon-processor state, including shadow state, is minimal, and is therefore maintained by a host CPU.

Graphics system features

Pan and zoom are implemented as high-speed pixel-copy operations. Copy performance varies from four to 16 million pixels per second as a function of pixel zoom factor. A 1/4-screen zoom by a factor of two, for example, runs at seven frames per second. Smaller areas, common in window-capable systems, easily zoom at 30 frames per second. Because the effects of pan and zoom are limited to a single window—or to multiple windows with independent factors if desired—the full screen remains a useful resource.

An optional frame-grab board captures both NTSC and PAL video in real time. Images are transferred to an arbitrary window at the rate of 16 million pixels per second. Once in raster memory, these images are operated on and displayed just like geometry-based images. Multiple buffers in the frame-grab hardware ensure that synchronization is correct and that neither images nor performance is lost.

The Image Engines support *alpha blending,* allowing pixel color components to be replaced with a linear combination of their previous components and the incoming color components. Application programmers select a specific blending equation from a list of options (see the section on the raster subsys-

tem). Perhaps the most commonly used blending function is

$$C_{dst} = \alpha_{src} C_{src} + (1-\alpha_{src}) C_{dst}$$

When α_{src} represents opacity of the incoming pixel value, the pixel is rendered transparently. Alternatively, when α_{src} represents pixel coverage (perhaps computed by the antialiased line-drawing hardware) the edges of the object being drawn are smoothed.

Antialiased lines, critical to molecular modeling applications, are drawn at up to 1/2 the rate of standard lines. Pixel coverage is computed from the fractional line position in the direction perpendicular to its primary direction (see Figure 8). The computed coverage, in the range 0...1, is used to scale the α of each pixel in the line.

$$\alpha_{src} = \alpha_{coverage} \alpha_{src}$$

A blending function such as the one described above is used to merge the line into the frame buffer.

Summary

The 4D/240GTX represents state-of-the-art in superworkstation architecture. Its parallel-scalar CPU architecture supports an attractive performance balance of 80 MIPS and 16 Mflops. Graphics performance of over 100,000 lighted, z-buffered quadrilaterals per second is provided by its tightly integrated graphics subsystem. A complete set of development software—including a parallelizing Fortran compiler, user-level parallelizing directives, and the immediate mode graphics library—allows application programs to achieve optimal performance easily. Its GT-XSD architecture ensures that full CPU and graphics performance are available simultaneously to application programmers. ■

References

1. R. Swanson and L. Thayer, "A Fast Shaded-Polygon Renderer," *Computer Graphics* (Proc. SIGGRAPH), Vol. 20, No. 4, Aug. 1986, pp. 95-101.
2. J. Torborg, "A Parallel Processor Architecture for Graphics Arithmetic Operations," *Computer Graphics* (Proc. SIGGRAPH), Vol. 21, No. 4, July 1987, pp. 197-204.
3. B. Apgar, B. Bersack, and A. Mammen, "A Display System for the Stellar Graphics Supercomputer Model GS1000," *Computer Graphics* (Proc. SIGGRAPH), Vol. 22, No. 4, Aug. 1988, pp. 255-262.
4. C. Thacker et al., "Alto: A Personal Computer," in *Computer Structures: Readings and Examples*, D. Siewiorek, G. Bell, and A. Newell, eds., McGraw-Hill, New York, 1982, pp. 548-572.
5. T. Myer and I. Sutherland, "On the Design of Display Processors," *CACM*, Vol. 11, No. 6, June 1968.
6. W. Tiller, "Rational B-Splines for Curve and Surface Representation," *CG&A*, Vol. 3, No. 6, Sept. 1983, pp. 61-69.
7. I. M. Papamarcos and J. Patel, "A Low-Overhead Coherence Solution for Multiprocessors with Private Cache Memories," *Proc. 11th Int'l Symp. Computer Architecture*, CS Press, Los Alamitos, Calif., 1984, pp. 348-354.
8. J. Dongarra, "Performance of Various Computers Using Standard Linear Equations in a Fortran Environment," Argonne Nat'l Lab., Feb. 16, 1988.
9. K. Akeley and T. Jermoluk, "High-Performance Polygon Rendering," *Computer Graphics* (Proc. SIGGRAPH), Vol. 22, No. 4, Aug. 1988, pp. 239-246.
10. H. Gouraud, "Continuous Shading of Curved Surfaces," *IEEE Trans. Computers*, Vol. 20, No. 6, June 1971, pp. 623-629.
11. T. Porter and T. Duff, "Compositing Digital Images," *Computer Graphics* (Proc. SIGGRAPH), Vol. 18, No. 3, July 1984, pp. 253-259.

Kurt Akeley is chief engineer in the Advanced Systems Division of Silicon Graphics Computer Systems. He has been with Silicon Graphics since its founding in 1982 and has contributed to the development of both CPU and graphics subsystems. His interests include hardware systems design, high-performance graphics, and schematic and simulation tool development.

Akeley received a BEE from the University of Delaware in 1980 and an MS in electrical engineering from Stanford University in 1982. He is a member of both IEEE and ACM SIGGRAPH.

Akeley can be contacted at Silicon Graphics Computer Systems, 2011 N. Shoreline Blvd., Mountain View, CA 94039-7311.